B3

PAT-NO: JP02001216140A

DOCUMENT-IDENTIFIER: JP 2001216140 A

TITLE: DEVICE FOR ALLOCATING INSTRUCTION CACHE FUNCTION, METHOD FOR OPTIMIZING ALLOCATION AND RECORDING MEDIUM RECORDING ALLOCATION OPTIMIZING PROCEDURE

PUBN-DATE: August 10, 2001

INVENTOR-INFORMATION:

NAME	COUNTRY
NIMATA, HIDEJI	N/A

ASSIGNEE-INFORMATION:

NAME	COUNTRY
NEC MICROSYSTEMS LTD	N/A

APPL-NO: JP2000027218

APPL-DATE: January 31, 2000

INT-CL (IPC): G06F009/06, G06F009/45 , G06F012/08

ABSTRACT:

PROBLEM TO BE SOLVED: To obtain an instruction cache function allocation device capable of reducing cache conflict between plural functions, and improving performing speed of an application program.

SOLUTION: A function time series information output part 100 inputs an application program 110 and outputs accessed function ID obtained at the time of accessing the function by a profile and returned function ID obtained at the time of return from the function to time series information

111 of function
performance. A function time series information analysis part 101 analyzes the
time series information 111, extracts a function accessing pattern having the
possibility of generating direct/indirect cache conflict and records the
extracted pattern as inter-function cache conflict combination information 112.
A function memory space arrangement optimizing part 102 arranges a memory space
in order from a function having the highest execution transition frequency so
that a cache line is not shared.

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2001－216140
（P2001－216140A）
(43)公開日　平成13年８月10日(2001.8.10)

(51)Int.Cl.⁷		識別記号	FI		テーマコード(参考)
Ｇ０６Ｆ	9/06	４１０	Ｇ０６Ｆ	9/06	４１０Ｈ　５Ｂ００５
	9/45			12/08	Ｓ　５Ｂ０７６
	12/08				Ｗ　５Ｂ０８１
				9/44	３２２Ｈ

審査請求　有　　請求項の数８　ＯＬ　（全 12 頁）

(21)出願番号　特願2000－27218(P2000－27218)

(22)出願日　平成12年１月31日(2000.1.31)

(71)出願人　000232036
エネイーシーマイクロシステム株式会社
神奈川県川崎市中原区小杉町１丁目403番53

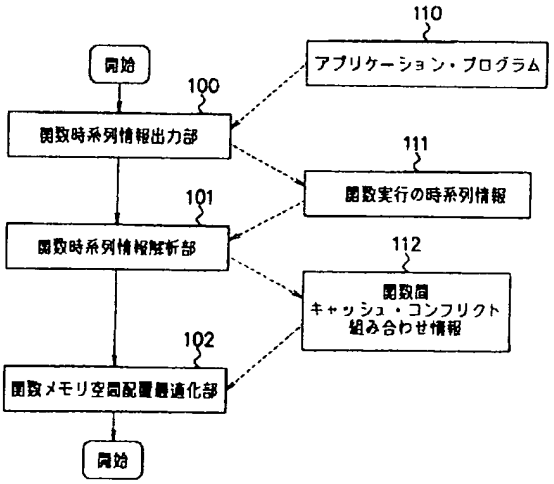
(72)発明者　二俣　秀治
神奈川県川崎市中原区小杉町一丁目403番53　日本電気アイシーマイコンシステム株式会社内

(74)代理人　100084250
弁理士　丸山　隆夫

Ｆターム(参考)　5B005 JJ13 LL04 MM02
5B076 AB02 BA01
5B081 CC27

(54)【発明の名称】　命令キャッシュ関数割付装置、割付最適化方法および割付最適化手順を記録した記録媒体

(57)【要約】
【課題】　複数の関数間のキャッシュ・コンフリクトを削減し、アプリケーション・プログラムの実行スピードを向上する命令キャッシュ関数割付装置を得る。
【解決手段】　関数時系列情報出力部１００がアプリケーション・プログラム１１０を入力してプロファイルにより関数呼び出し時の呼び出し先関数ＩＤと関数からの戻り時の戻り先関数ＩＤを関数実行の時系列情報１１１に出力し、この時系列情報１１１を関数時系列情報解析部１０１が解析し、直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出して関数間キャッシュ・コンフリクト組合せ情報１１２として記録する。関数メモリ空間配置最適化部１０２は、実行遷移回数の多い関数の順にキャッシュ・ラインを共有しないようにメモリ空間に配置する。



1

【特許請求の範囲】

【請求項1】　所定のアプリケーション・プログラムを入力してプロファイルにより関数呼び出し時の呼び出し先関数ＩＤおよび該関数から戻り時の戻り先関数ＩＤを出力する関数時系列情報出力部と、

前記関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、該抽出したパターンの関数の組合せと直接の関数呼出し関係にある関数の組合せおよびそれらの実行遷移回数を記録する関数時系列情報解析部と、

を有して構成されたことを特徴とする命令キャッシュ関数割付装置。

【請求項2】　前記関数時系列情報出力部から出力された呼び出し先関数ＩＤと戻り先関数ＩＤとを関数実行の時系列情報として蓄積する記憶部をさらに有することを特徴とする請求項1記載の命令キャッシュ関数割付装置。

【請求項3】　前記関数の組合せおよびそれらの実行遷移回数を関数間キャッシュ・コンフリクト組合せ情報として蓄積する記憶部を、さらに有することを特徴とする請求項1または2に記載の命令キャッシュ関数割付装置。

【請求項4】　プロファイルにより関数実行の時系列情報を出力し、

前記時系列情報から連続した関数呼出し等直接の関数呼出し以外にキャッシュ・コンフリクトを発生する可能性のある関数の組合せ実行パターンの関数間キャッシュ・コンフリクト組合せ情報を検出し、

前記検出した関数間キャッシュ・コンフリクト組合せ情報に対して関数の中で複数の関数が連続して呼ばれている場合あるいはループの中で呼ばれている場合等のこれら複数の関数間のキャッシュ・コンフリクトを削減し、所定のアプリケーション・プログラムの実行スピードを向上したことを特徴とする命令キャッシュ関数割付最適化方法。

【請求項5】　前記関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、該抽出したパターンの関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回数を前記関数間キャッシュ・コンフリクト組合せ情報に記録することを特徴とする請求項4に記載の命令キャッシュ関数割付最適化方法。

【請求項6】　前記関数間キャッシュ・コンフリクト組合せ情報は、キャッシュ・コンフリクトを発生する可能性のある組合せの2つの関数ＩＤを登録する2つの関数ＩＤ欄と該2つの組合せの関数実行遷移回数を設定する実行遷移回数欄とを備えていることを特徴とする請求項4または5に記載の命令キャッシュ関数割付最適化方法。

2

【請求項7】　プロファイルにより関数実行の時系列情報を出力する手順と、

前記時系列情報から連続した関数呼出し等直接の関数呼出し以外にキャッシュ・コンフリクトを発生する可能性のある関数の組合せ実行パターンの関数間キャッシュ・コンフリクト組合せ情報を検出する手順と、

前記検出した関数間キャッシュ・コンフリクト組合せ情報に対して関数の中で複数の関数が連続して呼ばれている場合あるいはループの中で呼ばれている場合等のこれら複数の関数間のキャッシュ・コンフリクトを削減する手順とを実行させるためのプログラムを記録した記録媒体。

【請求項8】　前記関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、該抽出したパターンの関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回数を前記関数間キャッシュ・コンフリクト組合せ情報に記録することを特徴とする請求項7に記載のプログラムを記録した記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、命令キャッシュへの関数割り付けを最適化する命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体に関する。

【0002】

【従来の技術】従来、命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体は、例えば、ＣＰＵの速度に規制されつつ、メモリアクセス処理速度を改善する装置および方法に関する。これに関連する技術において、ＣＰＵ速度とメモリアクセスの処理速度との差は、年々拡大する一方である。

【0003】ＤＲＡＭのアクセス時間の向上率は年率7%、ＣＰＵ速度は50～100%というデータもある（1996年4月19日／日経ＢＰ社発行、日経ＢＰ出版センター発売の「コンピュータの構成と設計」、David.A.Patterson、John.L.Hennessy、成田光彰訳より）、よって、キャッシュの有効利用が、非常に重要な問題となってきている。なお、キャッシュとは既知の通り、外部メモリを高速にアクセスするための機構である。一般に、プログラムはメモリに配置された関数をアドレス順に処理していく。しかし、外部メモリのアクセス速度は、ＣＰＵの処理速度に比べて非常に遅い、このため、結果として実行速度が遅くなるという問題点がある。

【0004】上記の問題点を解消するため、関数実行の

際に外部メモリ上のプログラムをキャッシュという高速アクセス可能なバッファにコピーして実行することにより、高速なプログラム実行を実現することが可能となる。これは一般に、プログラムを実行すると、一度アクセスされたメモリは近いうちに再度アクセスされる可能性が高いという性質によるものである。

【０００５】ただし、一般的にキャッシュの構成には、コストがかかるため外部メモリに比べて非常にサイズは小さい。よって、外部メモリをキャッシュのサイズで区切った領域に分割し、分割した領域毎に外部メモリをキャッシュに割り当て、あるアドレスに対する最初のアクセスでそのアドレスのプログラムをキャッシュの割り当てた境域にコピーし、次に外部メモリの同じアドレスをアクセスした場合は、キャッシュを直接アクセスする。このことで、高速なプログラム実行を実現する。

【０００６】この時、外部メモリからキャッシュ・メモリへのコピーは特定のサイズの単位で行われ、このサイズでキャッシュを分割した領域をキャッシュラインと呼ぶ。従って、同一のキャッシュラインに割り当たった外部メモリのアドレスに配置された関数同士は、関数が切り替わる度にキャッシュにプログラムをコピーし直す必要が生じてくる。これをキャッシュ・コンフリクトという。これが頻繁に起きると、結果としてプログラムの実行速度が遅くなってしまうという問題がある。よって、昨今では、この問題を解消すべく同時に動く可能性の高い関数同士は、同一のキャッシュラインには載らないように配置する方法が研究されている。

【０００７】なお、従来、キャッシュには命令キャッシュとデータキャッシュがあるが、本発明は、命令キャッシュに着目するものである。外部メモリのキャッシュへの割当て方式には、最も単純で安価なダイレクト・マップ方式や、セット・アソシアティブ／フル・アソシアティブといった方式がある。しかし、基本的な問題は全て同じであるため、以降、ダイレクト・マップ方式を例にとり説明する。

【０００８】また、従来の言語処理系プログラムでは、ある処理単位（関数）毎にメモリに適当に配置していた。このため、キャッシュ搭載のシステムにおいては、それが必ずしも最適に利用されているとは限らなかった。

【０００９】昨今では、命令キャッシュに関し、キャッシュ・コンフリクトの回数を確率的に減らすために、関数の呼び出し回数情報を元にして関数のメモリ配置を最適化する。このことにより、キャッシュを有効に利用する研究がなされており、いくつかのアルゴリズムが論文発表されている。例えば、従来例１の“Efficient Procedure Mapping Using Cache Line Colorling”(A.H.Hashemi,D.R.Kaeli,B.Calder ACM SIGPLAN,June,1997)においては、呼び出し回数の多い関数から順にキャッシュ・コンフリクトを避けるように関数のメモリ配置を最適

にしていく手法が公開されている。

【００１０】また、従来例２の特開平１１－２３２１１７号公報の「プログラム変換方法、プログラム変換装置及びプログラム変換プログラムを記憶した記憶媒体」においては、キャッシュ・メモリを効率よく使用するための関数配置方法の実施例として上記従来例１の論文の方法が採用され、詳しく説明されている。この論文のアルゴリズムでは、関数の呼出グラフを作成し、呼び出し回数をその辺に対する重みとして優先順位を付けてメモリ空間に配置する。このことにより、まず関数を最初に配置した時のキャッシュ・コンフリクトを避けることができる。さらに、各関数が配置された「色」（＝使用されているキャッシュライン）と、その関数が現在利用できない「色」の集合を記録しておき、後者の色を使わないように関数を配置し、既に配置した関数についても、その関数の利用できない色を使わないという条件の下で、別の場所に移動する。このことで、直接の“親”あるいは“子”との間で発生するキャッシュ・コンフリクトを除去するものである。

【００１１】図８、図９、および図１０を参照し、図３に示す本発明の実施例で適用したソース・プログラムに対する従来例での動作を確認する。このことにより、従来例における問題点をより詳細に説明する。なお、図８および図９において、実線の矢印は処理の流れを示し、点線の矢印はデータの流れを示している。

【００１２】図８の従来技術例に対し、図３に示すソース・プログラム例３００を適用した場合、関数呼び出し情報出力部８００において、プロファイルにより関数呼び出し時に呼出元と呼出先の関数情報とその呼出回数を出力すると、図１０に示す関数呼出組合せ情報８１１が作成される。

【００１３】次に、関数メモリ空間配置最適化部８０１において、関数呼出組合せ情報８１１を呼出回数の多い順にソートし、この順番にアドレス空間に配置すると同時に配置した関数が利用できない「色」の集合を認識し、これを避けて後続の関数を配置する。

【００１４】すなわち、図９において、ステップ９０１では関数呼出組合せ情報８１１から図１１に示す関数呼出しグラフ１１００を作成し、ステップ９０２では作成した関数呼出グラフ１１００を呼び出し回数の多いものと少ないものに分割する。ここでは、func-funcA、func-funcB、func-funcCが前者の「多いもの」、main-funcが後者の「少ないもの」となる。ここで作成した呼出グラフにおいては、関数の組合せが辺となり、その両端のノードが組合せにおける２つの関数となる。なお、図３における各関数の占めるキャッシュライン数すなわち「色」の数は、funcが２個、funcA,funcB,funkC,mainが１個であるものとする。

【００１５】次に、ステップ９０３において、呼出回数の多いもののグループを呼出回数の多い順にソートし、

その順番でステップ９０４以降の処理を行う。ステップ９０４では呼出回数の多い辺が残っているか確認し、残っているのでステップ９０５に進み、func-funcAの辺に対して両側のノードが未配置であるかを確認する。この確認において未配置であるのでステップ９０９に進み、funcとfuncＡをメモリ空間上の任意の場所に隣接して配置し、ステップ９１５においてfuncとfuncＡの利用できない「色」を利用不可能集合として認識した後、再びステップ９０４に戻る。隣接して配置されたfunc-funcAの辺は、複合ノードとして今後ひとつのノードとして扱われる。この時点で、既に配置済みの関数とキャッシュ・ラインの関係および各関数の利用不可能集合の状態は、図１２ａ）に示す様になっている。

【００１６】続いて、呼出回数の多い辺がまだ残っているのでステップ９０５に進み、func-funcCの辺に対して両側のノードが未配置であるかを確認する。この確認において、funcは配置であるのでステップ９０６に進み、２個の異なる複合ノードに属するノードを結ぶ辺かどうかを確認する。この確認において、funcＣは複合ノードに属していないので、ステップ９０７に進み、一方のノードが複合ノードに属し他方のノードが未配置かどうか確認すると、条件に当てはまるのでステップ９１１に進む。

【００１７】ステップ９１１では未配置のfuncＣをfuncに近い場所に配置し、ステップ９１３において関数配置の際、利用不可能集合の影響で隙間が空いてないか確認すると空いていないので、ステップ９１５においてfuncとfuncＣを利用できない「色」を利用不可能集合として認識した後に、再びステップ９０４に戻る。この時点で、既に配置済みの関数とキャッシュ・ラインの関係および各関数の利用不可能集合の状態は、図１２ｂ）に示す様になっている。

【００１８】ステップ９０４において、未だ未配置の辺があるのでステップ９０５に進み、func-funcBの辺に対して両側のノードが未配置であるかを確認する。この確認において、funcは配置済みであるのでステップ９０６に進み、２個の異なる複合ノードに属するノードを結ぶ辺かどうかを確認する。この確認において、funcＢは複合ノードに属していないのでステップ９０７に進み、一方のノードが複合ノードに属し、他方のノードが未配置かどうか確認すると条件に当てはまるのでステップ９１１に進む。

【００１９】ステップ９１１では、未配置のfuncＢと対を成すfuncの中心から複合ノードの両端までの距離が同じである。このため任意に左側に配置し、ステップ９１３において関数配置の際に利用不可能集合の影響で隙間が空いてないかを確認する。この認識の結果は空いていないので、ステップ９１５においてfuncとfuncＢの利用できない「色」を利用不可能集合として認識した後、再びステップ９０４に戻る、ステップ９０４において、未配置の辺が無くなったことを確認するとステップ９１６に進み、ただ１つの未配置ノードmainを、任意にfuncＡの右に配置する。

【００２０】図１２ｃ）は、最終的な関数配置とキャッシュ・ラインの関係、および各関数の利用不可能集合の状態であるが、funcＡ、funcＢが同じキャッシュ・ライン「青」を共有しており、それぞれの利用不可能集合には「青」が含まれていない。よって、呼び出し元関数と呼び出し先関数の間のキャッシュ・コンフリクトは削減できる。

【００２１】上記の従来例２では、手続き、関数、あるいはサブルーチン同士が、互いを呼び出す際のキャッシュ・メモリ上での衝突およびキャッシュ・ミスを防止するのが目的である。この目的において、手続き、関数、あるいはサブルーチンが実際に呼び出される回数を示す情報と、手続き、関数、あるいはサブルーチン同士が、互いを呼び出す関係を示す情報とを利用している。これにより、手続き、関数、あるいはサブルーチン同士が、互いを呼び出す際のキャッシュ・メモリ上での衝突を防止できる。

【００２２】

【発明が解決しようとする課題】しかしながら、上記の従来例２において、ある関数の中で複数の関数が連続して呼ばれている場合、あるいはループの中で呼ばれている場合等には、これら複数の関数間のキャッシュ・コンフリクトを削減できるとは限らないという問題点がある。

【００２３】従来例２との比較において、本発明の目的は、互いの呼び出しに関係の無い手続き、関数あるいはサブルーチンがある手続き、関数あるいはサブルーチンの中で複数の手続き、関数あるいはサブルーチンが図３に示すプログラム例のfuncＡ、funcＢ、funcＣのように連続して呼ばれている場合、あるいはループの中で呼ばれている場合に生じるキャッシュ・メモリ上での衝突およびキャッシュ・ミスを、防止することにある。

【００２４】また、本発明の構成では、手続き、関数、サブルーチンの時系列情報を利用し、その時系列情報をパターン認識のアルゴリズムによる解析により抽出したキャッシュ・メモリ上での衝突およびキャッシュ・ミスを発生する可能性のある手続き、関数、あるいはサブルーチンの組合せ情報を利用している。この構成により、上記従来例２の効果に加え、互いの呼び出し関係に無い手続き、関数、あるいはサブルーチン同士のキャッシュ・メモリ上での衝突およびキャッシュ・ミスの発生をも、防止することができる。

【００２５】本発明は、複数の関数間のキャッシュ・コンフリクトを削減し、アプリケーション・プログラムの実行スピードを向上する命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体を提供するこ

とを目的とする。

【００２６】より詳細には、本発明は、プロファイルにより直接の関数呼出組合せ情報８１１を出力する替わりに、関数実行の時系列情報１１１を出力し、この時系列情報から、連続した関数呼出し等直接の関数呼出し以外にキャッシュ・コンフリクトを発生する可能性のある関数の組合せ実行パターンを検出し、検出した関数間キャッシュ・コンフリクト組合せ情報１１２に対して、従来技術の関数配置最適化８０１を適用する。これにより従来削減できなかった、ある関数の中で複数の関数が連続して呼ばれている場合、あるいはループの中で呼ばれている場合等、これら複数の関数間のキャッシュ・コンフリクトを削減し、アプリケーション・プログラムの実行スピードを向上することを目的とする。

【００２７】

【課題を解決するための手段】かかる目的を達成するため、請求項１記載の発明の命令キャッシュ関数割付装置は、所定のアプリケーション・プログラムを入力してプロファイルにより関数呼び出し時の呼び出し先関数ＩＤおよび関数から戻り時の戻り先関数ＩＤを出力する関数時系列情報出力部と、関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、この抽出したパターンの関数の組合せと直接の関数呼出し関係にある関数の組合せおよびそれらの実行遷移回数を記録する関数時系列情報解析部と、を有して構成されたことを特徴としている。

【００２８】また、上記の関数時系列情報出力部から出力された呼び出し先関数ＩＤと戻り先関数ＩＤとを関数実行の時系列情報として蓄積する記憶部と、この関数の組合せおよびそれらの実行遷移回数を関数間キャッシュ・コンフリクト組合せ情報として蓄積する記憶部をさらに有するとよい。

【００２９】請求項４記載の発明の命令キャッシュ関数割付最適化方法は、プロファイルにより関数実行の時系列情報を出力し、この時系列情報から連続した関数呼出し等直接の関数呼出し以外にキャッシュ・コンフリクトを発生する可能性のある関数の組合せ実行パターンの関数間キャッシュ・コンフリクト組合せ情報を検出し、検出した関数間キャッシュ・コンフリクト組合せ情報に対して関数の中で複数の関数が連続して呼ばれている場合あるいはループの中で呼ばれている場合等のこれら複数の関数間のキャッシュ・コンフリクトを削減し、所定のアプリケーション・プログラムの実行スピードを向上したことを特徴としている。

【００３０】また、上記の関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、この抽出したパターンの関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回数を関数間キャッシュ・コンフリクト組合せ情報に記録し、さらに、この関数間キャッシュ・コンフリクト組合せ情報は、キャッシュ・コンフリクトを発生する可能性のある組合せの２つの関数ＩＤを登録する２つの関数ＩＤ欄とこの２つの組合せの関数実行遷移回数を設定する実行遷移回数欄とを備えるとよい。

【００３１】請求項７に記載の発明の命令キャッシュ関数割付最適化手順を記録した記録媒体は、プロファイルにより関数実行の時系列情報を出力する手順と、時系列情報から連続した関数呼出し等直接の関数呼出し以外にキャッシュ・コンフリクトを発生する可能性のある関数の組合せ実行パターンの関数間キャッシュ・コンフリクト組合せ情報を検出する手順と、検出した関数間キャッシュ・コンフリクト組合せ情報に対して関数の中で複数の関数が連続して呼ばれている場合あるいはループの中で呼ばれている場合等のこれら複数の関数間のキャッシュ・コンフリクトを削減する手順とを実行させることを特徴としている。

【００３２】また、上記の関数実行の時系列情報を解析して直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、この抽出したパターンの関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回数を関数間キャッシュ・コンフリクト組合せ情報を記録するとよい。

【００３３】

【発明の実施の形態】次に、添付図面を参照して本発明による命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体の実施の形態を詳細に説明する。図１から図７を参照すると、本発明の命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体の一実施形態が示されている。

【００３４】本発明の特徴は、キャッシュを搭載したマイクロ・プロセッシング・デバイス・システム用アプリケーション・プログラムについて、プロファイルにより得られた関数単位の実行に関する時系列情報を解析する。特に、直接の呼び出し関係を持たない関数同士の実行において、キャッシュのコンフリクトを生じる可能性のある関数の実行パターンを検出し、これらの関数同士が実行時に同じキャッシュラインを共有しないようなメモリ空間上のアドレスに配置する。このことにより、プログラムの実行スピードを向上させる。この内容を以下に詳述する。

【００３５】図１は、本発明の命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体の実施形態の処理手順例を示すブロック図である。また、図２は、関数間キャッシュ・コンフリクト組合せ情報１１２

の構成例を示す図である。なお、図１において、実線の矢印は処理の流れを示し、点線の矢印はデータの流れを示している。

【００３６】関数時系列情報出力部１００は、アプリケーション・プログラム１１０を入力し、プロファイルにより関数呼び出し時の呼び出し先関数ＩＤと関数からの戻り時の戻り先関数ＩＤを、関数実行の時系列情報１１１に出力する。

【００３７】関数時系列情報解析部１０１は、関数実行の時系列情報１１１を解析し、直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、関数間キャッシュ・コンフリクト組合せ情報１１２に、抽出したパターンの関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回数を記録する。

【００３８】関数メモリ空間配置最適化部１０２は、実行遷移回数の多い関数の順にキャッシュ・ラインを共有しないようにメモリ空間に配置する。また、実行遷移回数欄２０２は、関数ＩＤ欄２００、２０１とその組合せの関数実行遷移回数を設定する。

【００３９】関数間キャッシュ・コンフリクト組合せ情報１１２は、図２に示すように、キャッシュ・コンフリクトを発生する可能性のある組合せの２つの関数ＩＤを登録する。なお、関数ＩＤ欄２００、２０１とその組合せの関数実行遷移回数を設定する実行遷移回数欄２０２を備えている。

【００４０】上記に構成される本実施形態の命令キャッシュ関数割付装置において、まず、関数時系列情報出力部１００において入力したアプリケーション・プログラム１１０に対してプロファイルを行い、関数実行の時系列情報１１１を生成する。次に、関数時系列情報出力部１００において生成された関数実行の時系列情報１１１を解析し、同じキャッシュ・ラインを共有していた場合に、実行においてキャッシュ・コンフリクトを生じる関数単位の実行遷移パターンを認識し、関数間キャッシュ・コンフリクト組合せ情報１１２に、認識された関数の組合せとその実行遷移回数を記録する。最後に、関数メモリ空間配置最適化部１０２において、関数間キャッシュ・コンフリクト組合せ情報１１２を参照し、実行遷移回数の多い関数の順に同じキャッシュ・ラインを共有しないようにメモリ空間に配置する。

【００４１】以上の通り、プロファイルによる関数単位の実行に関する時系列情報を解析し、キャッシュのコンフリクトを生じる可能性のある関数の実行パターンを検出し、これらの関数同士が実行時に同じキャッシュ・ラインを共有しないようなメモリ空間上のアドレスに配置する。このことにより、プログラムの実行スピードを向上させることができる　これらの関係において、図１、図３、図４、図５、図６、および図７を参照して動作例について以下に詳述する。

【００４２】図１において、アプリケーション・プログラム１１０を入力した関数時系列情報出力部１００は、関数呼び出し時の呼び出し先関数ＩＤと関数からの戻り時に戻り先関数ＩＤを出力するコードを、アプリケーション・プログラム１１０に挿入して実行する。プロファイルと呼ぶこのことにより、これらの情報を関数実行の時系列情報１１１に出力する。

【００４３】すなわち、アプリケーション・プログラム１１０のＣ言語によるソース・プログラムが図３に示すソース・プログラム例３００の場合、関数funcの関数ＩＤを“０”、関数funcＡの関数ＩＤを“１”、関数funcＢの関数ＩＤを“２”、関数funcＣの関数ＩＤを“３”、関数mainの関数ＩＤを“４”と仮定する。本仮定において、まず、関数mainから関数funcの呼出時に呼出先関数ＩＤの“０”を出力して、次に関数funcから関数funcＡの呼出時に“１”を出力し、続いて関数funcＡから関数funcへの戻り時に戻り先関数ＩＤの“０”を出力し、以降、同様にしてプログラムの終了まで関数ＩＤの列を出力する。

【００４４】ソース・プログラム例３００は、Ｃ言語で関数mainから関数funcを呼び出し、関数funcでは関数funcＡと関数funcＢを連続して呼び出す処理を１０回繰返す処理と、続いて関数funcＡと関数funcＣを連続して呼び出す処理を２０回繰返す処理を行い、関数mainに戻って終了するプログラムである。この処理により、プログラムの終了までプロファイルを実行すると、図４に示す関数ＩＤの並びである関数時系列情報例４００が作成される。

【００４５】次に、関数時系列情報解析部１０１で、関数時系列情報出力部１００で作成した関数実行の時系列情報１１１を解析し、直接的、間接的なキャッシュ・コンフリクトを発生する可能性のある関数呼び出しのパターンを抽出し、関数間キャッシュ・コンフリクト組合せ情報１１２に抽出したパターンの関数の組合せと、直接の関数呼出し関係にある関数の組合せ、およびそれらの実行遷移回数を記録する。

【００４６】すなわち、図５を参照して関数時系列情報解析部１０１の詳細を説明する。なお、本図５において、実線の矢印は処理の流れを示し、点線の矢印はデータの流れを示している。まず、関数ＩＤ入力手段５００において、関数時系列情報１１１から全ての関数ＩＤの並びを入力する．この入力順に関数ＩＤに“１”からの番号を付け、番号変数初期化処理５０１によって番号変数ｎを“１”に初期化する．

【００４７】次に、関数時系列情報の終端判定処理５０２によってｎ＋１が入力した関数ＩＤの並びの総数を超えないと判定すると、再帰呼び出し判定処理５０３に進み、ｎとｎ＋１の関数ＩＤが“０”と“１”で異なるので再帰呼び出しではないと判定する．この判定の後、関数直接呼出し組合せ既登録判定処理５０４によってｎと

ｎ＋１の関数ＩＤの組合せが、関数間キャッシュ・コンフリクト組合せ情報１１２内に未だ存在しないと判定して、関数直接呼出し組合せ登録処理５０５によってｎとｎ＋１の関数ＩＤ（０，１）の組合せを関数間キャッシュ・コンフリクト組合せ情報１１２に登録し、実行遷移回数欄に“１”を設定する。以上の処理によって、図６のａ）に示す関数間キャッシュ・コンフリクト組合せ情報１１２が作成される。

【００４８】さらに、認識パターン関数ＩＤ確認処理５０６によってｎ＋４が関数ＩＤの総数を超えないことを確認すると、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０７において、ｎ、ｎ＋２、ｎ＋４の関数ＩＤは“０”であり一致することを認識したと判定し、第二の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０８において、ｎ＋１、ｎ＋３の関数ＩＤが“１”と“２”であり一致しないことを認識したと判定する。

【００４９】次に、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０７の判定により、ｎ＋２はｎと同一の関数ＩＤであり、ｎ＋１とｎ＋２は先の関数直接呼出し組合せ登録処理５０５によって登録済みのfuncとfuncＡの組合せであることが自明となる。このため、関数戻り組合せ登録処理５０９によってｎ＋１とｎ＋２、すなわちｎの関数ＩＤ（０，１）の組合せの実行遷移回数に“１”を加算し、図６のｂ）に示す関数間キャッシュ・コンフリクト組合せ情報１１２が作成される。

【００５０】続いて、コンフリクト関数組合せ既登録判定処理５１０によって、ｎ＋１とｎ＋３の関数ＩＤである“１”と“２”の組合せが関数間キャッシュ・コンフリクト組合せ情報１１２内に未だ存在しないと判定の後、コンフリクト関数組合せ登録処理５１１によって、ｎ＋１とｎ＋３の関数ＩＤ（１，２）の組合せを関数間キャッシュ・コンフリクト組合せ情報１１２に登録して実行遷移回数欄に１を設定し、さらに、次の関数時系列情報のパターン認識処理を行う。このため、番号更新処理５１２によって、ｎすなわち“０”をｎ＋２すなわち“２”に更新し、関数時系列情報の終端判定処理５０２に戻る。この時点で、図６のｃ）に示す関数間キャッシュ・コンフリクト組合せ情報１１２が作成されている。

【００５１】再び、関数時系列情報の終端判定処理５０２によってｎ＋１が入力した関数ＩＤの並びの総数を超えないと判定すると、再帰呼び出し判定処理５０３に進み、ｎとｎ＋１の関数ＩＤが“０”と“２”で異なるので再帰呼び出しではないと判定する。この後、関数直接呼出し組合せ既登録判定処理５０４によって、ｎとｎ＋１の関数ＩＤの組合せが関数間キャッシュ・コンフリクト組合せ情報１１２内に未だ存在しないと判定し、関数直接呼出し組合せ登録処理５０５によって、ｎとｎ＋１の関数ＩＤ（０，２）の組合せを関数間キャッシュ・コンフリクト組合せ情報１１２に登録し、実行遷移回数欄に“１”を設定する。以上の処理によって、図６のｄ）に示す関数間キャッシュ・コンフリクト組合せ情報１１２が作成される。

【００５２】さらに、認識パターン関数ＩＤ確認処理５０６によって、ｎ＋４が関数ＩＤの総数を超えないことを確認すると、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０７において、ｎ、ｎ＋２、ｎ＋４の関数ＩＤは“０”であり一致することを認識したと判定し、第二の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０８において、ｎ＋１、ｎ＋３の関数ＩＤが“２”と“１”であり一致しないことを認識したと判定する。

【００５３】次に、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０７の判定により、ｎ＋２はｎと同じ関数ＩＤであり、ｎ＋１とｎ＋２は先の関数直接呼出し組合せ登録処理５０５によって登録済みのfuncとfuncＢの組合せであることが自明である。このため、関数戻り組合せ登録処理５０９によって、ｎ＋１とｎ＋２、すなわちｎの関数ＩＤ（１，２）の組合せの実行遷移回数に“１”を加算する。

【００５４】続いて、コンフリクト関数組合せ既登録判定処理５１０によって、ｎ＋１とｎ＋３の関数ＩＤである“２”と“１”の組合せが関数間キャッシュ・コンフリクト組合せ情報１１２に既に存在すると判定し、実行遷移回数更新処理５１５によって当該組合せの実行遷移回数に“１”を加算し、さらに、次の関数時系列情報のパターン認識処理を行う。このため、番号更新処理５１２によって、ｎすなわち“２”をｎ＋２すなわち“４”に更新し、関数時系列情報の終端判定処理５０２に戻る。この時点で、図６のｅ）に示す関数間キャッシュ・コンフリクト組合せ情報１１２が作成されている。

【００５５】以降、図４に示す関数ＩＤの並びに対し、ｎが“４０”になるまで同様の処理を繰り返すと、図６のｆ）に示す関数間キャッシュ・コンフリクト組合せ情報１１２が作成される。

【００５６】さらに、図４に示す関数ＩＤの並びである関数時系列情報例４００に対して処理を進めると、この関数時系列情報例４００の終端判定処理５０２によってｎ＋１が関数ＩＤの並びの総数を超えないと判定すると、再帰呼び出し判定処理５０３に進む。ステップ５０３の「ｎとｎ＋１の関数ＩＤは異なるか？」の判定において、ｎすなわち“４０”とｎ＋１すなわち“４１”の関数ＩＤが“０”と“１”で異なるので再起呼び出しではないと判定し、関数直接呼出し組合せ既登録判定処理５０４によってｎとｎ＋１の関数ＩＤの組合せが関数間キャッシュ・コンフリクト組合せ情報１１２内に既に存在すると判定して、実行遷移回数更新処理５１４によって当該組合せの実行遷移回数に“１”を加算する。

【００５７】さらに、認識パターン関数ＩＤ確認処理５

０６によって、ｎ＋４が関数ＩＤの総数を超えないことを確認すると、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０７において、ｎ、ｎ＋２、ｎ＋４の関数ＩＤが“０”であり一致することを認識したと判定し、第二の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０８において、ｎ＋１、ｎ＋３の関数ＩＤが“１”と“３”であり一致しないことを認識したと判定する。

【００５８】次に、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０７の判定により、ｎ＋２はｎと同一の関数ＩＤであり、ｎ＋１とｎ＋２は先の関数直接呼出し組合せ登録処理５０５によって、登録済みのfuncとfuncＡの組合せであることが自明となる。このため、関数戻り組合せ登録処理５０９によってｎ＋１とｎ＋２、すなわちｎの関数ＩＤ（１，３）の組合せの実行遷移回数に“１”を加算する。

【００５９】続いて、コンフリクト関数組合せ既登録判定処理５１０によって、ｎ＋１とｎ＋３の関数ＩＤである“１”と“３”の組合せが関数間キャッシュ・コンフリクト組合せ情報１１２にまだ存在しないと判定し、コンフリクト関数組合せ登録処理５１１によってｎ＋１とｎ＋３の関数ＩＤの組合せを関数間キャッシュ・コンフリクト組合せ情報１１２に登録し、実行遷移回数欄に“１”を設定する、さらに、次の関数時系列情報のパターン認識処理を行うため、番号更新処理５１２によってｎすなわち“４０”をｎ＋２すなわち“４２”に更新し、関数時系列情報の終端判定処理５０２に戻る。この時点で、図６のｇ）に示す関数間キャッシュ・コンフリクト組合せ情報１１２が作成される。

【００６０】再び、関数時系列情報の終端判定処理５０２によって、ｎ＋１が入力した関数ＩＤの並びの総数を超えないと判定すると再帰呼び出し判定処理５０３に進み、ｎとｎ＋１の関数ＩＤが“０”と“３”で異なるので再起呼び出しではないと判定する。すると、関数直接呼出し組合せ既登録判定処理５０４によって、ｎとｎ＋１の関数ＩＤの組合せが関数間キャッシュ・コンフリクト組合せ情報１１２内にまだ存在しないと判定して、関数直接呼出し組合せ登録処理５０５によって、ｎとｎ＋１の関数ＩＤ（０，３）の組合せを関数間キャッシュ・コンフリクト組合せ情報１１２に登録し、実行遷移回数欄に“１”を設定する。

【００６１】さらに、認識パターン関数ＩＤ確認処理５０６によって、ｎ＋４が関数ＩＤの総数を超えないことを確認すると、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０７において、ｎ、ｎ＋２、ｎ＋４の関数ＩＤが“０”であり一致することを認識したと判定する。この後、第二の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０８において、ｎ＋１、ｎ＋３の関数ＩＤが“３”と“１”であり、一致しないことを認識したと判定する。

【００６２】次に、第一の間接的関数間キャッシュ・コンフリクト組合せパターン認識判定処理５０７の判定により、ｎ＋２はｎと同じ関数ＩＤであり、ｎ＋１とｎ＋２は先の関数直接呼出し組合せ登録処理５０５によって登録済みのfuncとfuncＣの組合せであることが自明である。このため、関数戻り組合せ登録処理５０９によって、ｎ＋１とｎ＋２、すなわちｎの関数ＩＤ（０，３）の組合せの実行遷移回数に“１”を加算する。

【００６３】続いて、コンフリクト関数組合せ既登録判定処理５１０によって、ｎ＋１とｎ＋３の関数ＩＤである“３”と“１”の組合せが関数間キャッシュ・コンフリクト組合せ情報１１２内に既に存在すると判定し、実行遷移回数更新処理５１５によって当該組合せの実行遷移回数に“１”を加算する。さらに、次の関数時系列情報がパターンに合致するか確認するため、番号更新処理５１０によって、ｎすなわち“４２”をｎ＋２すなわち“４４”に更新し、関数時系列情報の終端判定処理５０２に戻る。この時点で、図６のｈ）に示す関数間キャッシュ・コンフリクト組合せ情報１１２が作成されている。

【００６４】以降、図４に示す関数ＩＤの並びである関数時系列情報例４００に対し、ｎが“１２０”になるまで同様の処理を繰り返すと、図６のｉ）に示す関数間キャッシュ・コンフリクト組合せ情報１１２が作成される。

【００６５】さらに、図４に示す関数ＩＤの並びである関数時系列情報例４００に対し、処理を進めると、関数時系列情報の終端判定処理５０２によってｎ＋１が関数ＩＤの並びの総数を超えないと判定する。すると、再帰呼び出し判定処理５０３に進み、ｎとｎ＋１の関数ＩＤが“０”と“４”で異なるので再起呼出ではないと判定し、関数直接呼出し組合せ既登録判定処理５０４によって、ｎとｎ＋１の関数ＩＤの組合せが関数間キャッシュ・コンフリクト組合せ情報１１２内にまだ存在しないと判定する。この後、関数直接呼出し組合せ登録処理５０５によって、ｎとｎ＋１の関数ＩＤ（０，４）の組合せを関数間キャッシュ・コンフリクト組合せ情報１１２に登録し、実行遷移回数欄に“１”を設定する。

【００６６】さらに、認識パターン関数ＩＤ確認処理５０６によって、ｎ＋４が“１２４”であり、関数ＩＤの総数１２１を超えていることを確認すると、番号更新処理５１３に進み、ｎすなわち“１２０”をｎ＋１すなわち“１２１”に更新して関数時系列情報の終端判定処理５０２に戻り、ｎ＋１が関数ＩＤの総数に超えたと判定され、関数時系列情報解析部１０１の処理を終了する。この時点で、図６のｊ）に示す関数間キャッシュ・コンフリクト組合せ情報１１２が、作成されている。

【００６７】最後に、関数メモリ空間配置最適化部１０３で、図９に示す従来技術と同じ処理を、従来の関数組

合せ情報８１１の替わりに関数間キャッシュ・コンフリクト組合せ情報１１２を入力として行う。この後、関数間キャッシュ・コンフリクト組合せ情報にfuncAとfuncBの組合せが含まれることにより、図７に示す最終的な関数配置とキャッシュ・ラインの関係、および各関数の利用不可能集合の状態にみられるように、funcA、funcBが同じキャッシュ・ラインを共有すること無く配置される。

【００６８】尚、上述の実施形態は本発明の好適な実施の一例である。但し、これに限定されるものではなく、本発明の要旨を逸脱しない範囲内において種々変形実施が可能である。

【００６９】

【発明の効果】以上の説明より明らかなように、本発明の命令キャッシュ関数割付装置では、関数時系列情報から間接的キャッシュ・コンフリクトを発生する可能性のある関数の呼び出しパターンを抽出し、直接の関数呼び出しによるキャッシュ・コンフリクトを発生する可能性のある関数呼び出しパターンと合わせて、その関数実行遷移回数の多い順に同じキャッシュ・ラインを共有しないようにメモリ空間上に配置する。この直接、間接的なキャッシュ・コンフリクトを極力発生しないようにメモリ空間上に配置することにより、アプリケーション・プログラムの実行スピードを向上することができる。

【図面の簡単な説明】

【図１】本発明の命令キャッシュ関数割付装置、命令キャッシュ関数割付最適化方法および命令キャッシュ関数割付最適化手順を記録した記録媒体の実施形態の処理手順例を示すブロック図である。

【図２】関数間キャッシュ・コンフリクト組合せ情報１１２の構成例を示す図である。

【図３】ソース・プログラム例３００を示す図である。

【図４】関数時系列情報例４００を示す図である。

【図５】関数時系列情報解析部１０１の処理手順例を示すフローチャートである。

【図６】関数間キャッシュ・コンフリクト組合せ情報例１１２を示す図である。

【図７】最終的な関数配置とキャッシュ・ラインの関係、および各関数の利用不可能集合の状態を示す図である。

【図８】従来の命令キャッシュ関数割付装置の処理手順例を示すブロック図である。

【図９】従来の関数メモリ空間配置最適化部８０１の処理手順例を示すフローチャートである。

【図１０】従来の関数呼出組合せ情報８１１の構成例を示す図である。

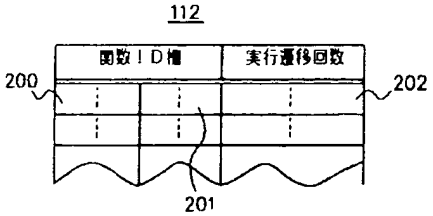
【図１１】従来の関数呼出グラフ１１００の構成を説明するための図である。

【図１２】従来の関数配置とキャッシュ・ラインの関係および各関数の利用不可能集合の状態を示す図である。

【符号の説明】

１００　関数時系列情報出力部
１０１　関数時系列情報解析部
１０２　関数メモリ空間配置最適化部
１０３　関数メモリ空間配置最適化部
１１０　アプリケーション・プログラム
１１１　関数実行の時系列情報
１１２　関数間キャッシュ・コンフリクト組合せ情報
２００、２０１　関数ＩＤ欄
２０２　実行遷移回数欄
３００　ソース・プログラム例
５００　関数ＩＤ入力手段
５０１　番号変数初期化処理
５０２　関数時系列情報の終端判定処理
５０３　再帰呼び出し判定処理
５０４　関数直接呼出し組合せ既登録判定処理

【図２】



【図３】

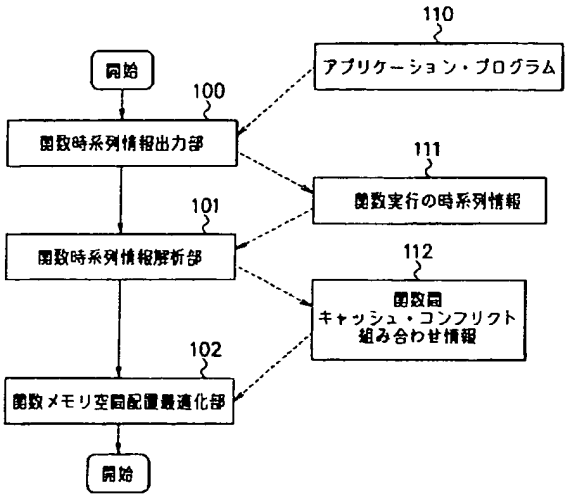
```
300
void func 0 ----------0
{   int  i;
    for (i=0; i<10; i++) {
        funcA 0 ----------1
        funcB 0 ----------2
    }
    for (i=0; i<20; i++) {
        funcA 0 ; --------1
        funcC 0 ; --------3
    }
}

void main 0 ----------4
{
    func 0; ----------0
}
```

【図１０】

呼出し元	呼出し先	実行遷移回数
main	func	1
func	funcA	30
func	funcB	10
func	funcC	20

【図1】

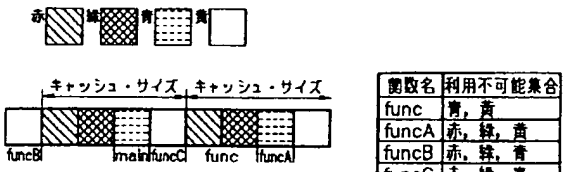


【図4】

400

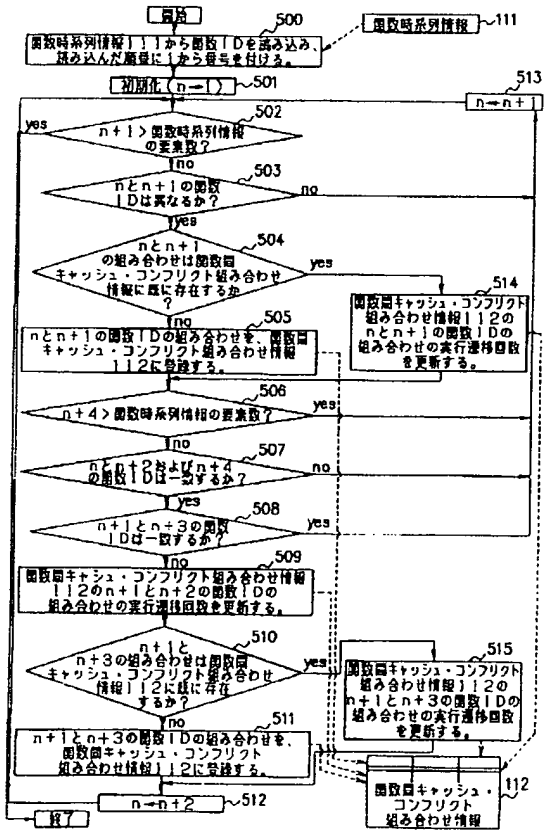
0, 1, 0, 2, （以降同じ並びの9回繰返し）, 一次行に続く
0, 1, 0, 3, （以降同じ並びの19回繰返し）, 4

【図7】



関数名	利用不可能集合
func	青，黄
funcA	赤，緑，黄
funcB	赤，緑，青
funcC	赤，緑，青
main	赤，緑

【図5】



【図6】

a)

組み合わせ		実行遷移回数
func (0)	funcA (1)	1

b)

組み合わせ		実行遷移回数
func	funcA (1)	2

c)

組み合わせ		実行遷移回数
func (0)	funcA (1)	2
funcA (1)	funcB (2)	1

d)

組み合わせ		実行遷移回数
func (0)	funcA (1)	2
funcA (1)	funcB (2)	1
func (0)	funcB (2)	1

e)

組み合わせ		実行遷移回数
func (0)	funcA (1)	2
funcA (1)	funcB (2)	2
func (0)	funcB (2)	2

f)

組み合わせ		実行遷移回数
func (0)	funcA (1)	20
funcA (1)	funcB (2)	20
func (0)	funcB (2)	20

g)

組み合わせ		実行遷移回数
func (0)	funcA (1)	21
funcA (1)	funcB (2)	20
func (0)	funcB (2)	20
funcA (1)	funcC (3)	1

h)

組み合わせ		実行遷移回数
func (0)	funcA (1)	21
funcA (1)	funcB (2)	20
func (0)	funcB (2)	20
funcA (1)	funcC (3)	2
func (0)	funcC (3)	2

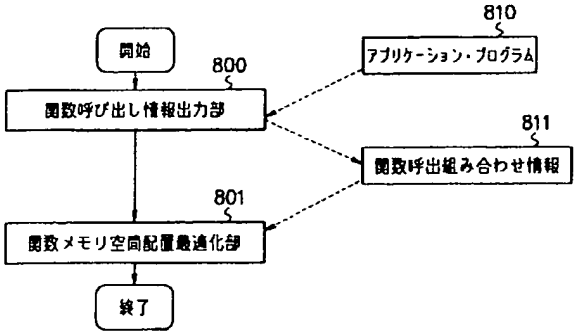
i)

組み合わせ		実行遷移回数
func (0)	funcA (1)	60
funcA (1)	funcB (2)	20
func (0)	funcB (2)	20
funcA (1)	funcC (3)	38
func (0)	funcC (3)	39

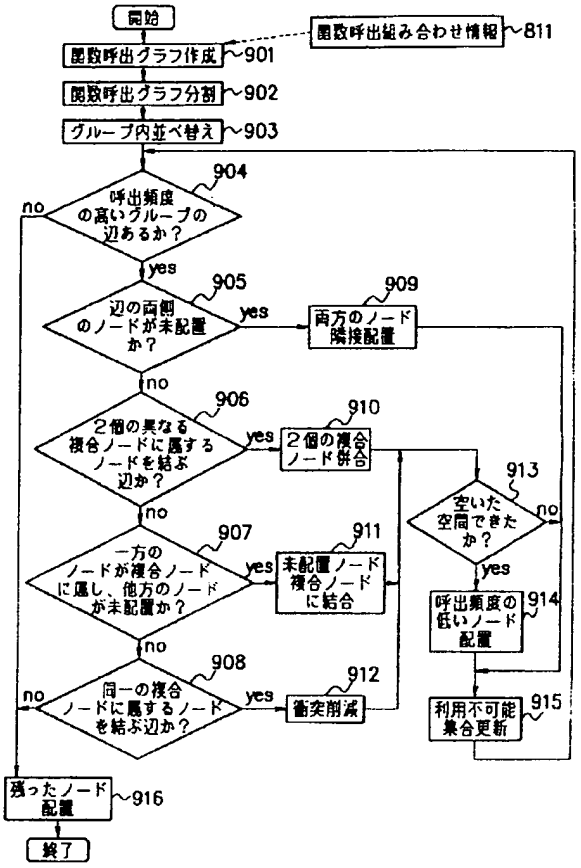
j)

組み合わせ		実行遷移回数
func (0)	funcA (1)	60
funcA (1)	funcB (2)	20
func (0)	funcB (2)	20
funcA (1)	funcC (3)	38
func (0)	funcC (3)	39
func (0)	main (4)	1

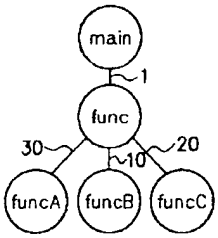
【図8】



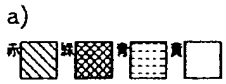
【図9】

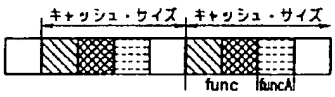


【図11】

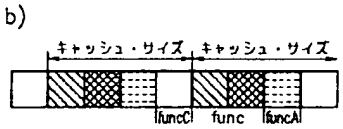


【図12】

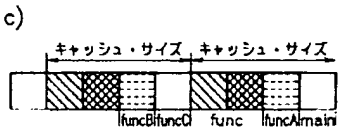




関数名	利用不可能集合
func	青，黄
funcA	赤，緑
funcB	未配置
funcC	未配置
main	未配置



関数名	利用不可能集合
func	青，黄
funcA	赤，緑
funcB	未配置
funcC	赤，緑
main	未配置



関数名	利用不可能集合
func	青，黄
funcA	赤，緑
funcB	赤，緑
funcC	赤，緑
main	赤，緑

【手続補正書】
【提出日】平成12年2月25日（2000．2．25）
【手続補正1】
【補正対象書類名】明細書
【補正対象項目名】発明の名称
【補正方法】変更
【補正内容】
【発明の名称】　命令キャッシュ関数割付装置、割付最適化方法および割付最適化手順を記録した記録媒体